作业五

16.1-2

对于活动选择问题，每次选择可以启动的**最后一个活动**，尽可能推迟当前活动的开始时间，从而为更多的活动留出空间。

贪心算法满足两个性质：贪心选择性质：选择启动最晚的活动不会影响最终的最优解。如果在当前活动集合中选择启动最晚的活动是正确的，那么我们可以用它来代替最优解中的其他活动，并且仍然保持最优；最优子结构性质：在将一个活动加入选择集合后，剩下的未处理活动仍然构成一个活动选择问题的子问题。因此，可以递归地应用相同的贪婪策略解决子问题。

假设活动集合为$A = \{a_1, a_2, \ldots, a_n\}$，活动已经按照开始时间 s_i从大到小排序。令贪婪算法选出的活动集合为 S_G，最优解为 S_{OPT}。

每次选择可以启动的最后一个活动 a_k，该活动与当前选择集合S_G 中的所有活动兼容，剩余的未处理活动构成的子问题仍然是一个活动选择问题，通过递归贪心选择，最终可以得到一个合法的活动集合。

假设 S_G不是最优解，存在一个最优解S_{OPT}，且 $|S_{OPT}| > |S_G|$。

考虑 S_G 和 S_{OPT} 的第一个不一致活动 a_i，即 $a_i \notin S_G$且 $a_i \in S_{OPT}$。由于 a_i 是 S_{OPT}中的一部分，且 S_G 的选择是基于启动最晚的活动，而a_i 的启动时间比 S_G 中的对应活动更早，因此可以替换S_{OPT}中的活动，构造出一个新的最优解，使得选择与 S_G 一致。这与假设 S_G 不是最优解矛盾。

16.1-3

假设有以下活动集合，活动 a_i 定义为(s_i, f_i)：

i	1	2	3	4	5	6
s_i	1	3	0	4	8	5
f_i	4	5	6	7	9	9
持续时间	3	2	6	3	1	4

活动集合按持续时间递增排序：

i	5	2	1	4	6	3
s_i	8	3	1	4	5	0
f_i	9	5	4	7	9	6
持续时间	1	2	3	3	4	6

在当前活动集合中，选择持续时间最短的兼容活动。选择的活动集合为 $\{a_2, a_5\}$，总数为2。实际最优解是 $\{a_1, a_4, a_5\}$，总数为3，该策略将失败。

16.2-1

要证明0-1背包具有贪心选择性质，即证明其最优解可以有每一步的子问题的最优选择得到。

假设有n个物品，价值分别为$v_1, v_2, \ldots v_n$，重量分别为$w_1, w_2, \ldots, w_n$，背包容量为W。存在一个n元向量$(X_1, X_2, \ldots, X_n)$，在$\Sigma_{i=1}^{n} W_i * X_i \leq W$的条件下背包总价值$V = \Sigma_{i=1}^{n} V_i * X_i$最大。物品首先按照$\frac{V_i}{W_i}$降序排列。从$C_{i,j}$中选择物品优先考虑前面的物品，且满足总容量不超过W时，X越接近1越好。

记$A_{i,j}$为最优解，原问题转化为求$A_{1,n}$，记k为第k次从C中选择物品进背包。

Ⅰ）当k=1时，满足贪心选择性质，即第一次选物品p进背包，且p=1。下面用反证法证明：

若p≠1，则p≥2。但在此情况下不能保证$A_{1,n}$最优。试考虑$W_1 = W$的情况下，另外一个解$A'_{1,n} = 1$的价值V'更大（因为$\frac{V_1}{W_1} > \frac{V_2}{W_2} > \ldots > \frac{V_n}{W_n}$）。既$A_{i,j}$不是最优解，产生矛盾。所以p=1。

Ⅱ）在满足条件1的情况下，假设$k \leq z$时，满足贪心选择性质。既前z（包括z）次从$C_{z,n}$中选择物品，都是优先考虑选择物品z，且在满足条件1的情况下，X_i 越接近1越好。

Ⅲ）在满足条件1的情况下，当k=z+1时，证明也满足贪心选择性质，既第k=z+1次选物品（z+1）。

先证明$A_{1,n}$的子问题$A_{z+1,n}$也具有最优性质：如果存在$C_{z+1,n}$中选择物品的子问题的解$A'_{1,n}$的总价值比$A_{z+1,n}$的总价值更大，那么$A'_{z+1,n}$与$A'_{1,z}$合并后的原问题的解$A'_{1,n}$的总价值比$A_{1,n}$的总价值更大。这与$A_{1,n}$是最优解矛盾。所以$A_{z+1,n}$也具有最优性质。

于是第k=z+1次选择物品等价于子问题$A_{z+1,n}$的第一次选择物品，又因为在Ⅱ）假设成立的情况下，$C_{1,n}$的前z个物品已经被选了，所以转换成$A_{z+1,n}$从$C_{z+1,n}$中选择第一个物品。根据Ⅰ），显然优先选择物品（z+1）。所以结论得证。

16.2-2

dp[i][j]: 前 i个物品在背包容量为 j 的条件下的最大总价值。

如果不选择第 i 个物品：dp[i][j] = dp[i-1][j]

如果选择第 i 个物品且j \geq w_i：dp[i][j] = \max(dp[i-1][j], dp[i-1][j-w_i] + v_i)

边界条件: dp[0][j] = 0 for all j（没有物品时总价值为0）; dp[i][0] = 0 for all i（容量为0时总价值为0）。

目标是计算dp[n][W]，即前 n个物品在容量 W下的最大价值

```
def knapsack_01(weights, values, W):
    n = len(weights)
    dp = [[0] * (W + 1) for _ in range(n + 1)]
    for i in range(1, n + 1):
        for j in range(W + 1):
            if weights[i - 1] <= j:
                dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weights[i - 1]]
                + values[i - 1])
            else:
                dp[i][j] = dp[i - 1][j]
    max_value = dp[n][W]
    included_items = []
    j = W
    for i in range(n, 0, -1):
        if dp[i][j] != dp[i - 1][j]:
            included_items.append(i - 1)
            j -= weights[i - 1]
    return max_value, included_items
```

16.2-3

在 0-1 背包问题中，若物品按 **权重递增排序** 和 **价值递减排序** 的顺序相同，则说明每个物品的价值和权重存在某种一致性。对任意两个物品 i 和 j，如果$w_i < w_j$，那么 $v_i > v_j$。这种特殊情况下，价值和权重呈反相关，可以优先选择“重量小且价值高”的物品。找到总重量不超过背包容量 W的最大总价值。由于这种排列顺序具备单调性，可以利用贪心算法直接找到最佳解，而无需动态规划。

按物品的顺序遍历，从第一个物品开始装入背包。若当前物品可以装入背包（即剩余容量足够），则选择该物品并更新剩余容量。如果当前物品的重量超过剩余容量，则直接停止（因为后续物品的权重更大）。

```
输入: 物品列表 (weights, values) 按权重递增排序, 背包容量 W
输出: 最大价值 max_value, 被选择的物品列表 selected_items

初始化: max_value = 0, remaining_capacity = W, selected_items = []
遍历每个物品 (w[i], v[i]):
    如果 w[i] <= remaining_capacity:
        选择物品 i
        更新 max_value += v[i]
        更新 remaining_capacity -= w[i]
        将 i 加入 selected_items
    否则:
        结束遍历
返回 max_value, selected_items
```

1. **贪心选择性质**:按权重递增且价值递减的顺序选择物品，相当于每次优先选择单位重量价值高的物品。由于按排序条件，选择顺序中的物品总是当前最优的。
2. **最优子结构**:剩余容量 W' 对应的子问题仍然是一个规模较小的同类问题。贪心策略在每一步选择当前最优解，最终构成全局最优解。
3. **排序顺序保障了不需要回溯**:后续物品的权重更大，且单位重量价值更低，不可能再改进解。
4. 总时间复杂度: $O(n \log n)$。